backOffice

# How To

## (v0.2 – luglio 2020)

**Generalità**

Il pacchetto backOffice è composto dalle seguenti componenti:

* **Loader** Modulo che carica le tabelle a partire da un CSV;
* **Extractor** Modulo che estrae un csv a partire da una select;
* **SQLExecutor** Esecutore di comandi SQL (insert, update, procedure, ecc);
* **Mail** Modulo che invia mail;
* **FTP** Modulo che fa upload di file FTP;
* **ZIP** Modulo che comprime files;

Tutti i flussi, ad eccezione di **ZIP**, utilizzano un file di properties in input, di seguito un esempio di parametrizzazione:

**DBClass=**com.mysql.cj.jdbc.Driver

**Conn=**mysqlx://80.211.28.47:3306/bomelito?serverTimezone=UTC&useSSL=false

**Usr=**phpmyadmin

**Psw=**\*\*\*\*\*\*

**ConnectionVariable0**=SET sql\_mode = NO\_BACKSLASH\_ESCAPES

**mail.smtp.host=**out.virgilio.it

**mail.smtp.auth=**true

**mail.smtp.port=**587

**mail\_usr=**anonymous.guest@virgilio.it

**mail\_psw=**\*\*\*\*\*

**INSERT\_START\_ELAB=**INSERT INTO batch.sched\_elaborazione (id\_catena,id\_elab\_prec,id\_step,pgm\_name,elab\_name,elab\_mode,start\_elab,end\_elab,max\_error\_level,n\_record\_elab,n\_trans\_elab,return\_code,id\_elab) values (?,?,?,?,?,?,?,?,?,?,?,?,?)

**INSERT\_START\_ELAB\_NO\_ID\_ELAB=**INSERT INTO batch.sched\_elaborazione (id\_catena,id\_elab\_prec,id\_step,pgm\_name,elab\_name,elab\_mode,start\_elab,end\_elab,max\_error\_level,n\_record\_elab,n\_trans\_elab,return\_code) values (?,?,?,?,?,?,?,?,?,?,?,?)

**PREC\_ELAB=**SELECT n\_record\_elab,id\_elab\_prec,return\_code,end\_elab FROM batch.sched\_elaborazione WHERE pgm\_name = ? and elab\_name = ? order by end\_elab desc

**UPDATE\_END\_ELAB=**UPDATE batch.sched\_elaborazione SET END\_ELAB = ?, MAX\_ERROR\_LEVEL = ?, return\_code = ? WHERE ID\_ELAB = ?

**INSERT\_ELAB\_LOG=**INSERT INTO batch.sched\_elaborazione\_log (id\_elab,id\_catena,prog\_msg,tipo\_msg,liv\_attivo,msg,id\_step) values (?,?,?,?,?,?,?)

**UPDATE\_RECORDS=**UPDATE batch.sched\_elaborazione SET n\_record\_elab = ? WHERE ID\_ELAB = ?

Dove:

* **DBClass** contiene la libreria java per MySql utilizzata;
* **Conn** contiene la stringa di connessione al db, di seguito il dettaglio per l’esempio indicato:
  + mysqlx 🡪 stringa standard per MySql, a volte può essere “mysql” a seconda della libreria utilizzata;
  + 80.211.28.47 🡪 indirizzo che ospita il DB;
  + 3306 🡪 porta;
  + Bomelito 🡪 nome del DB;
  + ?serverTimezone=UTC&useSSL=false 🡪 variabili di connessione al DB;
* **Usr** contiene la user di connessione al DB;
* **Psw** contiene la password di connessione al DB;
* **ConnectionVariable0, ConnectionVariable1, ecc** contengono variabili utilizzate dal **Extracor,** nell’esempio viene specificato di non considerare i comandi come “\n” o “\r” quando si estraggono i dati dalla select;
* **mail.smtp.host** contiene l’host per inviare le e-mail (modulo **Mail**);
* **mail.smtp.auth** variabile true/false per le e-mail (modulo **Mail**);
* **mail.smtp.port** contiene la porta per inviare le e-mail (modulo **Mail**);
* **mail\_usr** contiene la user per la login alla casella e-mail (modulo **Mail**);
* **mail\_psw** contiene la password per la login alla casella e-mail (modulo **Mail**);
* **INSERT\_START\_ELAB** contiene la insert utilizzata per inserire l’inizio elaborazione;
* **INSERT\_START\_ELAB\_NO\_ID\_ELAB** contiene la insert utilizzata per inserire l’inizio elaborazione ma senza id\_elab in quanto autocalcolata;
* **PREC\_ELAB** query che cerca la precedente elaborazione;
* **UPDATE\_END\_ELAB** update che aggiorna la end elab dell’elaborazione;
* **INSERT\_ELAB\_LOG** contiene la insert del log di elaborazione;
* **UPDATE\_RECORDS** update che aggiorna quanti record sono stati elaborati;

Tutti i flussi, ad esclusione di **ZIP**, loggano su file e su tabelle sched\_elaborazione e sched\_elaborazione\_log.

Per parametrizzare il log su file sarà sufficiente andare nel file di log4j “log4j.properties” presente nella cartella resources e modificare la stringa presente alla variabile “log4j.appender.file.File=”.

**Modulo Loader**

Il modulo Loader riceve i seguenti parametri in input:

Parametro 1) Nome flusso 🡪 va inserito “Loader”;

Parametro 2) File di properties 🡪 file di properties comune a tutti i moduli tranne ZIP;

Parametro 3) File descrittore di tabella XML 🡪 File XML che descrive le tabelle;

Parametro 4) File di input CSV 🡪 file contenente le righe da inserire in tabella;

Parametro 5) Modalità lancio (N/R) 🡪 “N” se modalità normale o “R” se bisogna “riprendere” un’esecuzione terminata male;

Parametro 6) Nome tabella 🡪 tabella dove andare a fare i caricamenti, deve fare “match” con una presente nel file XML;

Parametro 7) Svuotamento tabella (S/N) 🡪 se impostato a “S” permette lo svuotamento della tabella prima dell’inserimento, tale funzionalità è utilizzabile solo se si lancia in modalità “N”;

Parametro 8) ID\_CATENA 🡪 valore riportato in tabella per schedulatore esterno;

Parametro 9) ID\_STEP 🡪 valore riportato in tabella per schedulatore esterno;

Parametro 10) ID\_ELAB (se non inserito verra' autogenerato) 🡪 id\_elab della tabella sched\_elaborazione: o automatico o generato dallo schedulatore esterno;

**Parametro 3) File descrittore di tabella XML**

Questa sezione descrive come è strutturato il file XML che descrive le tabelle, iniziamo dall’intestazione:

<?xml version="1.0" encoding="UTF-8"?>

[<backOfficeLoader>](file:///C:\Programmazione%20&%20co\backOffice\Tabs_Collection_minuscole.xml)

Oltre all’encoding, che è standard XML, il root tag per convenzione è “backOfficeLoader” ma l’applicazione non fa differenza se si inserisce un tag diverso (serve giusto un root tag per racchiudere tutti gli altri), per quanto riguarda le tabelle, invece, di seguito un esempio:

[<anagrafica\_coll Commit=”**1000**” CSVSeparator=”**;**”>](file:///C:\Programmazione%20&%20co\backOffice\Tabs_Collection_minuscole.xml)

<Cd\_CF seq=”**0**” type=”**STRING**”/>

<Descrizione seq=”**1**” type=”**STRING**”/>

<PartitaIva seq=”**2**” type=”**STRING**”/>

<CodiceFiscale seq=”**3**” type=”**STRING**”/>

</anagrafica\_coll>

Il tag iniziale rappresenta il nome della tabella (occhio MySql è case sensitive) i tag sottostanti sono i campi della tabella, **gli attributi accettati dal tag della tabella sono:**

* Commit Indica ogni quante righe inserite viene effettuato il commit;
* CSVSeparator Indica il separatore tra i campi del csv;

**Gli attributi accettati dal tag colonna sono:**

* Seq Indica la posizione della colonna nel csv, 0 based;
* Type Indica il tipo di dato, i dati accettati sono:
  + STRING Stringa alfanumerica;
  + INT Numerico intero;
  + DECIMAL Numerico con virgola;
  + DATE Data;
  + TIMESTAMP Data con ora minuti e secondi;
* Format Da utilizzare solo con DECIMAL, DATE e TIMESTAMP:
  + Se DECIMAL Le formattazioni sono quelle standard Java come ad esempio “en-US” per rappresentare i decimali con il punto invece che con la virgola;
  + Se DATE Si possono usare “DD” per il giorno del mese, “MM” per il mese (numerico) dell’anno, “YYYY” per l’anno a 4 cifre e “YY” per l’anno a 2 cifre;
  + Se TIMESTAMP In aggiunta a date si possono usare HH (ore sempre in 24 ore), MI (minuti), SS (secondi);

**Le colonne, poi accettano, anche valori statici tra tag, ad esempio:**

<a Seq="**0**" type="**Int**">**ID\_ELAB**</a>

<b Seq="**1**"type="**STRING**">**ARGUMENT\_0**</b>

In questo caso nella colonna con nome “a” verrà messo sempre l’ID\_Elab della tabella di elaborazione, e nella colonna “b” verrà messo l’argomento 0 (il primo) passato nella riga di comando, nell’esempio di seguito c’è evidenziato in giallo il parametro che verrebbe portato in tabella:

**java -jar backOffice.jar Loader ./bo\_server.properties ./Tabs\_Collection\_minuscole.xml ./ANAGRAFICA\_COLL.csv N anagrafica\_coll S 123 345**

Ovviamente è possibile usare ARGUMENT\_1, ARGUMENT\_2, ecc, inoltre, oltre alle parole chiave ID\_ELAB e ARGUMENT\_0, è possibile inserire un qualsiasi valore statico nel tag che verrà riportato identico in tabella, ad esempio:

<b Seq="**1**"type="**STRING**">**Questa stringa verrà riportata identica in tabella**</b>

**Modulo Extractor**

Il modulo Extractor riceve i seguenti parametri in input:

Parametro 1) Nome flusso 🡪 va inserito “Extractor”;

Parametro 2) File di properties 🡪 file di properties comune a tutti i moduli tranne ZIP;

Parametro 3) File XML 🡪 File contenente le select di estrazione;

Parametro 4) Identificativo tag XML 🡪 deve fare “match” con il tag definito al parametro 3

Parametro 5) File di output CSV 🡪 File CSV generato a valle della select;

Parametro 6) ID\_CATENA 🡪 valore riportato in tabella per schedulatore esterno;

Parametro 7) ID\_STEP 🡪 valore riportato in tabella per schedulatore esterno;

Parametro 8) ID\_ELAB (se non inserito verra' autogenerato) 🡪 id\_elab della tabella sched\_elaborazione: o automatico o generato dallo schedulatore esterno;

**Parametro 3) File XML**

Questa sezione descrive come è strutturato il file XML, iniziamo dall’intestazione:

<?xml version="1.0" encoding="UTF-8"?>

[<backOfficeExtractor>](file:///C:\Programmazione%20&%20co\backOffice\Tabs_Collection_minuscole.xml)

Oltre all’encoding che è standard XML il root tag per convenzione è “backOfficeExtractor” ma l’applicazione non fa differenza se si inserisce un tag diverso (serve giusto un root tag per racchiudere tutti gli altri), per quanto riguarda le select invece:

[<query\_tipo\_1>](file:///C:\Programmazione%20&%20co\backOffice\Extractor.xml)

[<query>](file:///C:\Programmazione%20&%20co\backOffice\Extractor.xml)

[<![CDATA[](file:///C:\\Programmazione%20&%20co\\backOffice\\Extractor.xml)

[SELECT CONCAT('#ARGUMENT\_0#;',IFNULL(a,''),';',IFNULL(b,''),';',IFNULL(c,''),';',IFNULL(DATE\_FORMAT(d, '%Y%m%d'),''),';',IFNULL(DATE\_FORMAT(e, '%Y/%m/%d %T'),'')) FROM tab\_prova](file:///C:\\Programmazione%20&%20co\\backOffice\\Extractor.xml)

[]]>](file:///C:\\Programmazione%20&%20co\\backOffice\\Extractor.xml)

</query>

</query\_tipo\_1>

Bisogna dare un identificativo alla query (in questo caso “query\_tipo\_1”) che fa “match” con il parametro 4 passato a riga di comando, poi un tag “query” obbligatorio e infine la query che va messa in “CDATA” per non avere problemi coi caratteri speciali, come si può notare i campi sono concatenati con “;” a livello di query e anche tutte le conversioni dei campi in stringa avviene nella query. Infine, è possibile inserire i parametri passati in input al jar con ARGUMENT\_0, ARGUMENT\_1, ecc come avviene per il Loader.

**Modulo SQLExecutor**

Il modulo SQLExecutor riceve i seguenti parametri in input:

Parametro 1) Nome flusso 🡪 va inserito “SQLExecutor”;

Parametro 2) File di properties 🡪 file di properties comune a tutti i moduli tranne ZIP;

Parametro 3) File XML 🡪 File contenente i comandi SQL da eseguire;

Parametro 4) Identificativo tag XML 🡪 deve fare “match” con il tag definito al parametro 3

Parametro 5) ID\_CATENA 🡪 valore riportato in tabella per schedulatore esterno;

Parametro 6) ID\_STEP 🡪 valore riportato in tabella per schedulatore esterno;

Parametro 7) ID\_ELAB (se non inserito verra' autogenerato) 🡪 id\_elab della tabella: o automatico o generato dallo schedulatore esterno;

**Parametro 3) File XML**

Questa sezione descrive come è strutturato il file XML, iniziamo dall’intestazione:

<?xml version="1.0" encoding="UTF-8"?>

[<backOfficeSQLExecutor>](file:///C:\Programmazione%20&%20co\backOffice\Tabs_Collection_minuscole.xml)

Oltre all’encoding che è standard XML il root tag per convenzione è “backOfficeSQLExecutor” ma l’applicazione non fa differenza se si inserisce un tag diverso (serve giusto un root tag per racchiudere tutti gli altri), per quanto riguarda i comandi SQL invece di seguito gli esempi:

[<SQLCommand\_tipo\_1>](file:///C:\Programmazione%20&%20co\backOffice\SQLExecutor.xml)

[<SQLCommand>](file:///C:\Programmazione%20&%20co\backOffice\SQLExecutor.xml)

[<![CDATA[update tab\_prova set b='Aggiornato da SQLExecutor :)']]>](file:///C:\Programmazione%20&%20co\backOffice\SQLExecutor.xml)

</SQLCommand>

</SQLCommand\_tipo\_1>

[<SQLCommand\_tipo\_2>](file:///C:\Programmazione%20&%20co\backOffice\SQLExecutor.xml)

[<SQLCommand>](file:///C:\Programmazione%20&%20co\backOffice\SQLExecutor.xml)

[<![CDATA[CALL `update\_tab\_prova` ('aggiornata da stored lanciata da SQLExecutor')]]>](file:///C:\Programmazione%20&%20co\backOffice\SQLExecutor.xml)

</SQLCommand>

</SQLCommand\_tipo\_2>

[<SQLCommand\_tipo\_3 RETURN="**Y**">](file:///C:\Programmazione%20&%20co\backOffice\SQLExecutor.xml)

[<SQLCommand>](file:///C:\Programmazione%20&%20co\backOffice\SQLExecutor.xml)

[<![CDATA[select `select\_tab\_prova` (1)]]>](file:///C:\Programmazione%20&%20co\backOffice\SQLExecutor.xml)

</SQLCommand>

</SQLCommand\_tipo\_3>

[<SQLCommand\_tipo\_4 RETURN="**Y**">](file:///C:\Programmazione%20&%20co\backOffice\SQLExecutor.xml)

[<SQLCommand>](file:///C:\Programmazione%20&%20co\backOffice\SQLExecutor.xml)

[<![CDATA[select `select\_tab\_prova\_all` ()]]>](file:///C:\Programmazione%20&%20co\backOffice\SQLExecutor.xml)

</SQLCommand>

</SQLCommand\_tipo\_4>

Ogni comando può avere un nome arbitrario, nel caso d’esempio sono stati creati i quattro comandi SQLCommand\_tipo\_1, SQLCommand\_tipo\_2, SQLCommand\_tipo\_3 e SQLCommand\_tipo\_4, e possono avere un attributo RETURN=”Y” utilizzato dalle stored function che devono restituire un valore, ogni comando avrà sempre un tag SQLCommand che racchiude l’SQL da eseguire (si consiglia sempre l’utilizzo di “CDATA” per non avere problemi di caratteri speciali). Come si può notare negli esempi il primo comando esegue una semplice update, il secondo comando chiama una stored procedure, il terzo e il quarto comando una stored function ma il terzo con parametro “1” il quarto senza parametri. L’esempio è pensato per funzionare, ovviamente, solo con MySql.

**Modulo Mail**

Il modulo Mail riceve i seguenti parametri in input:

Parametro 1) Nome flusso 🡪 va inserito “SQLExecutor”;

Parametro 2) File di properties 🡪 file di properties comune a tutti i moduli tranne ZIP;

Parametro 3) File XML 🡪 File contenente i templates delle e-mails;

Parametro 4) Identificativo tag XML 🡪 deve fare “match” con il tag definito al parametro 3

Parametro 5) ID\_CATENA 🡪 valore riportato in tabella per schedulatore esterno;

Parametro 6) ID\_STEP 🡪 valore riportato in tabella per schedulatore esterno;

Parametro 7..N) ID\_ELAB o Attachment 🡪 id\_elab della tabella: o automatico o generato dallo schedulatore esterno e/o la lista degli attachment da inserire nella mail;

**Parametro 3) File XML**

Questa sezione descrive come è strutturato il file XML, iniziamo dall’intestazione:

<?xml version="1.0" encoding="UTF-8"?>

[<backOfficeMail>](file:///C:\Programmazione%20&%20co\backOffice\Tabs_Collection_minuscole.xml)

Oltre all’encoding che è standard XML il root tag per convenzione è “backOfficeMail” ma l’applicazione non fa differenza se si inserisce un tag diverso (serve giusto un root tag per racchiudere tutti gli altri), per quanto riguarda i templates delle e-mail invece:

[<mail\_tipo\_1>](file:///C:\Programmazione%20&%20co\backOffice\mail.xml)

<from>[maildelmittente@dominio.it</from](mailto:maildelmittente@dominio.it%3c/from)>

<to>[maildestinatario1@dominio.it,maildestinatario2@dominio.com</to](mailto:maildestinatario1@dominio.it,maildestinatario2@dominio.com%3c/to)>

<cc/>

<bcc/>

<subject>Tipo\_1</subject>

[<body>](file:///C:\Programmazione%20&%20co\backOffice\mail.xml)

[<![CDATA[](file:///C:\\Programmazione%20&%20co\\backOffice\\mail.xml)

[<p>RECORDS in tab\_prova: §SELECT concat('<b>',count(\*),', quarto parametro: #ARGUMENT\_3#','</b>') FROM tab\_prova§<p>](file:///C:\\Programmazione%20&%20co\\backOffice\\mail.xml)

[]]>](file:///C:\\Programmazione%20&%20co\\backOffice\\mail.xml)

</body>

[</mail\_tipo\_1>](file:///C:\Programmazione%20&%20co\backOffice\mail.xml)

Come al solito “mail\_tipo\_1” deve fare “match“ con il quarto parametro, come si può notare ci sono tutti i campi classici di una mail, **attenzione: se gli account di posta sono più di uno bisogna separarli con la “,” e non con “;”**, nel body è possibile inserire testo statico, codice html, delle select, e gli “ARGUMENT” come negli altri moduli, la select deve essere racchiusa tra “§” e gli argument tra “#”.

**Modulo FTP**

Il modulo FTP riceve i seguenti parametri in input:

Parametro 1) Nome flusso 🡪 va inserito “FTP”;

Parametro 2) File di properties 🡪 file di properties comune a tutti i moduli tranne ZIP;

Parametro 3) File XML 🡪 File contenente i templates degli FTP;

Parametro 4) Identificativo tag XML 🡪 deve fare “match” con il tag definito al parametro 3

Parametro 5) ID\_CATENA 🡪 valore riportato in tabella per schedulatore esterno;

Parametro 6) ID\_STEP 🡪 valore riportato in tabella per schedulatore esterno;

Parametro 7..N) ID\_ELAB o File FTP da inviare 🡪 id\_elab della tabella: o automatico o generato dallo schedulatore esterno e/o la lista dei file da inviare via FTP;

**Parametro 3) File XML**

Questa sezione descrive come è strutturato il file XML, iniziamo dall’intestazione:

<?xml version="1.0" encoding="UTF-8"?>

[<backOfficeMail>](file:///C:\Programmazione%20&%20co\backOffice\Tabs_Collection_minuscole.xml)

Oltre all’encoding che è standard XML il root tag per convenzione è “backOfficeFTP” ma l’applicazione non fa differenza se si inserisce un tag diverso (serve giusto un root tag per racchiudere tutti gli altri), di seguito un esempio di parametrizzazione dei templates:

[<FTP\_tipo\_1>](file:///C:\Programmazione%20&%20co\backOffice\ftp.xml)

<Server>indirizzo.del.server</Server>

<Port>numero porta</Port>

<Usr>utente da inserire</Usr>

<Psw>password da inserire</Psw>

<RemoteWorkingDirectory>path remote dove depositare i file

</RemoteWorkingDirectory>

</FTP\_tipo\_1>

**Modulo ZIP**

Il modulo ZIP riceve i seguenti parametri in input:

Parametro 1) Nome flusso 🡪 va inserito “ZIP”;

Parametro 2) -o (file di output) 🡪 valore statico “-o”;

Parametro 3) Nome del file di output 🡪 nome del file zip generato in output;

Parametro 4) -i (files di input) 🡪 valore statico “-i”;

Parametro 5..N) Lista dei files di input 🡪 lista dei file di input di cui fare lo zip;

**Esempi di command line per il lancio dei flussi**

Caricamento di tabella “anagrafica\_coll” in modalità “N” con svuotamento della tabella prima di iniziare il caricamento “S”, ID\_CATENA=123, ID\_STEP=345 e ID\_ELAB autogenerato e quindi non specificato:

*java -jar backOffice.jar* ***Loader*** *./bo\_server.properties ./Tabs\_Collection\_minuscole.xml ./ANAGRAFICA\_COLL.csv N anagrafica\_coll S 123 345*

Esecuzione di comando SQL “SQLCommand\_tipo\_1” con ID\_CATENA=123, ID\_STEP=345 e ID\_ELAB=1000:

*java -jar backOffice.jar* ***SQLExecutor*** *./bo\_server.properties ./SQLExecutor.xml SQLCommand\_tipo\_1 123 345 1000*

Invio mail di tipo “mail\_tipo\_1”, ID\_CATENA=123, ID\_STEP=345 e ID\_ELAB autogenerato e quindi non specificato:

*java -jar backOffice.jar* ***Mail*** *./bo\_server.properties ./mail\_server.xml mail\_tipo\_1 123 345*

Invio file FTP di tipo “mail\_tipo\_1”, ID\_CATENA=123, ID\_STEP=345, file ftp da inviare “activation-1.1.1.wso2v2.jar” e ID\_ELAB autogenerato e quindi non specificato:

*java -jar backOffice.jar* ***FTP*** *./bo\_server.properties ./ftp2.xml FTP\_tipo\_1 123 345 activation-1.1.1.wso2v2.jar*

Estrazione di tipo “query\_tipo\_1” che genera un file di output “tab\_prova\_out.csv”, ID\_CATENA=123, ID\_STEP=345 e ID\_ELAB autogenerato e quindi non specificato:

*java -jar backOffice.jar Extractor ./bo\_server.properties ./Extractor.xml query\_tipo\_1 ./tab\_prova\_out.csv 123 345*

Creazione di un file ZIP che genera un file di output “-o ./test.zip” comprimendo i file di input “-i ./bo\_server.properties ./Tabs\_Collection.xml ./marcheSRL.csv”

*java -jar backOffice.jar* ***ZIP*** *-o ./test.zip -i ./bo\_server.properties ./Tabs\_Collection.xml ./marcheSRL.csv*

**Esempi di elaborazioni su tabelle sched\_elaborazione/sched\_elaborazione\_log**

Di seguito lo schema della tabella **sched\_elaborazione**:

| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| --- | --- | --- | --- | --- | --- |
| id\_elab | int(11) | NO | PRI |  | auto\_increment |
| id\_catena | int(11) | NO | PRI |  |  |
| id\_elab\_prec | int(11) | YES |  |  |  |
| id\_step | int(11) | NO | PRI |  |  |
| pgm\_name | varchar(4096) | NO |  |  |  |
| elab\_name | varchar(4096) | NO |  |  |  |
| elab\_mode | char(1) | NO |  |  |  |
| start\_elab | timestamp | YES |  |  |  |
| end\_elab | timestamp | YES |  |  |  |
| max\_error\_level | int(11) | NO |  |  |  |
| n\_record\_elab | int(11) | NO |  |  |  |
| n\_trans\_elab | int(11) | NO |  |  |  |
| return\_code | int(11) | YES |  |  |  |

Il campo evidenziato in giallo è collegato logicamente con omonimo campo in **sched\_elaborazione\_log**:

| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| --- | --- | --- | --- | --- | --- |
| id\_elab | int(11) | NO | PRI |  |  |
| id\_catena | int(11) | NO | PRI |  |  |
| id\_step | int(11) | YES |  |  |  |
| prog\_msg | int(11) | NO | PRI |  |  |
| tipo\_msg | varchar(20) | NO |  |  |  |
| time\_stamp | timestamp | NO |  | CURRENT\_TIMESTAMP | on update CURRENT\_TIMESTAMP |
| liv\_attivo | int(11) | NO |  |  |  |
| dato\_elab | varchar(4000) | YES |  |  |  |
| msg | varchar(4000) | NO |  |  |  |
| proc\_gen\_error | varchar(50) | YES |  |  |  |

Facciamo un esempio:

* Un operatore effettua un caricamento con il **Loader** che ha una riga errata nel file;
* Fa il rerun a **R** dopoaver effettuato la fix;
* Il caricamento va a buon fine;

Quindi effettuerebbe prima il seguente run a **N** della tabella **tab\_prova** con svuotamento della tabella:

java -jar backOffice.jar **Loader** ./bo\_server.properties ./tab\_prova.xml ./tab\_prova.csv **N** tab\_prova S 123 345

Una volta andato in errore verrebbe lanciato nuovamente il flusso a **R,** notareanche come sia obbligatorio che nel rerun a R, ovviamente, la tabella non può essere svuotata**:**

java -jar backOffice.jar **Loader** ./bo\_server.properties ./tab\_prova.xml ./tab\_prova.csv **N** tab\_prova N 123 345

Di seguito i due run in tabella:

| **id\_elab** | **id\_catena** | **id\_elab\_prec** | **id\_step** | **pgm\_name** | **elab\_name** | **elab\_mode** | **start\_elab** | **end\_elab** | **max\_error\_level** | **n\_record\_elab** | **n\_trans\_elab** | **return\_code** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 24 | 123 |  | 345 | backOffice.Loader | tab\_prova.csv | N | 2020-04-22 09:24:04 | 2020-04-22 09:24:04 | 3 | 0 | 1000 | 16 |
| 25 | 123 | 0 | 345 | backOffice.Loader | tab\_prova.csv | R | 2020-04-22 09:41:47 | 2020-04-22 09:41:47 | 1 | 3 | 1000 | 0 |

La prima riga andata in errore è evidenziata in rosso, la seconda in verde è stata elaborata con successo, come si può notare il **return\_code** della prima è a 16 proprio per indicare che c’è stato un errore durante il caricamento e il secondo è a 0 perché l’elaborazione è andata a buon fine.

Gli stati possibili del return\_code sono:

* Null 🡪 elaborazione in esecuzione;
* 0 🡪 elaborazione completata con successo;
* 16 🡪 elaborazione in errore;
* -8 🡪 elaborazione in errore nelle fasi preliminari (es.: esiste un altro flusso in esecuzione mio gemello e non posso partire);

Per quanto riguarda l’errore -8 ogni flusso capisce quale sia il proprio gemello in base a pgm\_name e elab\_name, queste due info cambiano a seconda del flusso:

* **Loader** 🡪 pgm\_name=”backOffice.Loader” e elab\_name=<<il nome del file csv>>;
* **Extractor 🡪** pgm\_name=”backOffice.Extractor” e elab\_name=<<il nome del file csv>>;
* **FTP 🡪** pgm\_name=”backOffice.FTP” e elab\_name=<<nome del tag xml del FTP da utilizzare>> (nel esempio più in alto “FTP\_tipo\_1”);
* **Mail 🡪** pgm\_name=”backOffice.Mail” e elab\_name=<<nome del tag xml della mail da utilizzare>>(nel esempio più in alto “mail\_tipo\_1”);
* **SQLExecutor 🡪** pgm\_name=”backOffice.SQLExecutor” e elab\_name=<<nome del file xml da utilizzare>>;
* **ZIP 🡪** non scrive in tabella;

Ritornando al log degli esempi di backOffice.Loader di prima, di seguito le righe su sched\_elaborazione\_log:

| **id\_elab** | **id\_catena** | **id\_step** | **prog\_msg** | **tipo\_msg** | **time\_stamp** | **liv\_attivo** | **dato\_elab** | **msg** | **proc\_gen\_error** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 24 | 123 | 345 | 1 | INFO | 2020-04-22 09:24:04 | 1 |  | Elaborazione iniziata |  |
| 24 | 123 | 345 | 2 | ERROR | 2020-04-22 09:24:04 | 3 |  | Errore durante l'inserimento massivo alla riga 2. Formato non valido alla colonna 5, campo=e, tipo=TIMESTAMP, formato=DD/MM/YYYY HH:MI:SS e valore letto in input=32/03/2019 04:05:06 - messaggio d'errore: Timestamp format must be yyyy-mm-dd hh:mm:ss[.fffffffff] |  |
| 24 | 123 | 345 | 3 | INFO | 2020-04-22 09:24:04 | 1 |  | Elaborazione completata in errore |  |
| 25 | 123 | 345 | 1 | INFO | 2020-04-22 09:41:47 | 1 |  | Elaborazione iniziata |  |
| 25 | 123 | 345 | 2 | INFO | 2020-04-22 09:41:47 | 1 |  | Eseguita commit FINALE dopo 3 |  |
| 25 | 123 | 345 | 3 | INFO | 2020-04-22 09:41:47 | 1 |  | Elaborazione completata con successo |  |